

莫队算法详解

本文翻译自[MO's Algorithm \(Query square root decomposition\)](#)，作者anudeep2011，发表日期为2014-12-28。由于最近碰到一些莫队算法的题目，找到的相关中文资料都比较简略，而这篇英语文章则讲解的比较详细，故翻译成中文与大家分享。由于本人水平有限，错误在所难免，请谅解。下面是译文。

我又发现了一个有用，有趣但网上资源非常少的话题。在写作之前，我做了一个小调查，令我惊讶的是，几乎所有的印度程序员都不知道该算法。学习这个很重要，事实上所有的codeforces红名程序员都使用这个算法，比如在div 1 C题和D题中。在一年半以前没有这方面的题目，但从那时起这类题目的数量就爆发了！我们可以期待这在未来的比赛中会有更多的这类题目。

莫队算法详解

问题描述

复杂度 $O(N^2)$ 的简单的解法

一个解决上述问题的算法及其正确性

对上述算法的复杂性证明 - $O(\sqrt{N} * N)$

上述算法的适用范围

习题和示例代码

问题描述

给定一个大小为N的数组，数组中所有元素的大小 $\leq N$ 。你需要回答M个查询。每个查询的形式是L, R。你需要回答在范围[L, R]中至少重复3次的数字的个数。

例如：数组为{1, 2, 3, 1, 1, 2, 1, 2, 3, 1}（索引从0开始）

查询：L=0, R=4。答案=1。在范围[L, R]中的值={1, 2, 3, 1, 1}，只有1是至少重复3次的。

查询：L=1, R=8。答案=2。在范围[L, R]中的值={2, 3, 1, 1, 2, 1, 2, 3}，1重复3遍并且2重复3次。至少重复3次的元素数目=答案=2。

复杂度 $O(N^2)$ 的简单的解法

对于每一个查询，从L至R循环，统计元素出现频率，报告答案。考虑M=N的情况，以下程序在最

坏的情况运行在 $O(n^2)$

```
for each query:
    answer = 0
    count[] = 0
    for i in {l..r}:
        count[array[i]]++
        if count[array[i]] == 3:
            answer++
```

对上述算法稍作修改。它仍然运行在 $O(n^2)$

```
add(position):
    count[array[position]]++
    if count[array[position]] == 3:
        answer++
```

```
remove(position):
    count[array[position]]--
    if count[array[position]] == 2:
        answer--
```

```
currentL = 0
currentR = 0
answer = 0
count[] = 0
for each query:
    // currentL 应当到 L, currentR 应当到 R
    while currentL < L:
        remove(currentL)
        currentL++
    while currentL > L:
        add(currentL)
        currentL--
    while currentR < R:
        add(currentR)
        currentR++
    while currentR > R:
        remove(currentR)
        currentR--
    output answer
```

最初我们总是从L至R循环，但现在我们从上一次查询的位置调整到当前的查询的位置。如果上一次的查询是L = 3, R = 10, 则我们在查询结束时有currentL=3、currentR=10。如果下一个查询是L = 5, R = 7, 则我们将currentL 移动到5, currentR 移动到7。
add 函数 意味着我们添加该位置的元素到当前集合内，并且更新相应的回答。

`remove` 函数意味着我们从当前集合内移除该位置的元素，并且更新相应的回答。

一个解决上述问题的算法及其正确性

莫队算法仅仅调整我们处理查询的顺序。我们得到了 M 个查询，我们将把查询以一个特定的顺序进行重新排序，然后处理它们。显然，这是一个离线算法。每个查询都有 L 和 R ，我们称呼其为“起点”和“终点”。让我们将给定的输入数组分为 \sqrt{N} 块。每一块的大小为 $N/\sqrt{N} = \sqrt{N}$ 。每个“起点”落入其中的一块。每个“终点”也落入其中的一块。

如果某查询的“起点”落在第 p 块中，则该查询属于第 p 块。该算法将处理第1块中的查询，然后处理第2块中的查询，等等，最后直到第 \sqrt{N} 块。我们已经有一个顺序、查询按照所在的块升序排列。可以有很多的查询属于同一块。

从现在开始，我会忽略（其它，译者注，所有括号内斜体同）所有的块，只关注我们如何询问和回答第1块。我们将对所有块做同样的事。（第1块中的）所有查询的“起点”属于第1块，但“终点”可以在包括第1块在内的任何块中。现在让我们按照 R 值升序的顺序重新排列这些查询。我们也在所有的块中做这个操作。（指每个块块内按 R 升序排列。）

最终的排序是怎样的？

所有的询问首先按照所在块的编号升序排列（所在块的编号是指询问的“起点”属于的块）。如果编号相同，则按 R 值升序排列。

例如考虑如下的询问，假设我们会有3个大小为3的块（0-2,3-5,6-8）：

{0, 3} {1, 7} {2, 8} {7, 8} {4, 8} {4, 4} {1, 2}

让我们先根据所在块的编号重新排列它们

{0, 3} {1, 7} {2, 8} {1, 2} (|) {4, 8} {4, 4} (|) {7, 8}

现在我们按照 R 的值重新排列

{1, 2} {0, 3} {1, 7} {2, 8} (|) {4, 4} {4, 8} (|) {7, 8}

现在我们使用与上一节所述相同的代码来解决这个问题。上述算法是正确，因为我们没有做任何改变，只是重新排列了查询的顺序。

对上述算法的复杂性证明 - $O(\sqrt{N} * N)$

我们完成了莫队算法，它只是一个重新排序。可怕的是它的运行时分析。原来，如果我们按照我上面指定的顺序，我们所写的 $O(N^2)$ 的代码运行在 $O(\sqrt{N} \times N)$ 时间复杂度上。可怕，这是正确的，仅仅是重新排序查询使我们把复杂度从 $O(N^2)$ 降低到 $O(\sqrt{N} \times N)$ ，而且也没有任何进一步的代码上的修改。好哇！我们将以 $O(\sqrt{N} \times N)$ 的复杂度AC。

看看我们上面的代码，所有查询的复杂性是由4个while循环决定的。前2个while循环可以表述为“左指针(*currentL*)的移动总量”，后2个while循环可以表述为“右指针(*currentR*)的移动总量”。这两者的和将是总复杂性。

最重要的。让我们先谈论右指针。对于每个块，查询是递增的顺序排序，所以右指针（currentR）按照递增的顺序移动。在下一个块的开始时，指针可能在extreme end（最右端？），将移动到下一个块中的最小的R处。这意味着对于一个给定的块，右指针移动的量是 $O(N)$ 。我们有 $O(\sqrt{N})$ 块，所以总共是 $O(N * \sqrt{N})$ 。太好了！

让我们看看左指针怎样移动。对于每个块，所有查询的左指针落在同一个块中，当我们从一个查询移动到另一个查询左指针会移动，但由于前一个L与当前的L在同一块中，此移动是 $O(\sqrt{N})$ （块大小）的。在每一块中左指针的移动总量是 $O(Q * \sqrt{N})$ ，Q是落在那个块的查询的数量。对于所有的块，总的复杂度为 $O(M * \sqrt{N})$ 。

就是这样，总复杂度为 $O((N + M) * \sqrt{N}) = O(N * \sqrt{N})$

上述算法的适用范围

如前所述，该算法是离线的，这意味着当我们被强制按照特定的顺序查询时，我们不能再使用它。这也意味着当有更新操作时我们不能用这个算法。不仅如此，一个重要的可能的局限性：我们应该能够编写add和remove函数。会有很多的情况下，add是平凡的（指复杂度 $O(1)$ ？），但remove不是。这样的一个例子就是我们要求区间内最大值。当我们添加的元素，我们可以跟踪最大值。但当我们删除元素则不是平凡的。不管怎样，在这种情况下，我们可以使用一个集合来添加元素，删除元素和报告最小值（作者想说最大值？）。在这种情况下，添加和删除操作都是 $O(\log N)$ （导致了 $O(N * \sqrt{N} * \log N)$ 的算法）。

在许多情况下，我们可以使用此算法。在一些情况下，我们也可以使用其它的数据结构，如线段树，但对于一些问题使用莫队算法的是必须的。让我们在下一节中讨论几个问题。

习题和示例代码

DQUERY – SPOJ: 区间内不同元素的数量=出现次数 ≥ 1 的元素个数，所以跟以上讨论的问题是一样的。

[点击此处查看示例代码](#)

注意：该代码提交会超时，加上更快的输入输出(传说中的输入输出挂？)就能AC。为了使代码整洁，去掉了快的输入输出。（评论中说将remove和add声明为inline内联函数就可以AC。）

Powerful array – CF Div1 D: 这道题是必须用莫队算法的例子。我找不到任何其它解法。CF Div1 D意味着这是一道难题。看看用了莫队算法就多简单了。你只需要修改上述代码中的add(), remove()函数。

GERALD07 – Codechef

GERALD3 – Codechef

Tree and Queries – CF Div1 D

Powerful Array – CF Div1 D

Jeff and Removing Periods – CF Div1 D

(省略结束语)
